Python. Семинар 5

Преподаватели: Дмитрий Косицин, Светлана Боярович и Анастасия Мицкевич

Задание 1. (0.3 балла). Реализуйте генератор **unique**, принимающий в качестве аргумента некоторый iterable, и возвращающий последовательно из него только уникальные элементы в том порядке, в котором они встретились.

Функцию сохраните в файле *iterators.py*.

Пример

```
expected = [1, 2, 3]
actual = unique([1, 2, 1, 3, 2])
assert expected == list(actual)
```

Задание 2. (0.3 балла). Реализуйте функцию **transpose**, которая транспонирует iterable вложенных iterable. Предполагайте, что количество элементов во всех вложенных iterable одинаково. Другими словами, транспонирует прямоугольный двухмерный массив.

Воспользуйтесь функциями из модуля **itertools** и *built-in* функциями. Использовать циклы не разрешается.

Функцию сохраните в файле *iterators.py*.

Пример

```
expected = [[1, 2], [-1, 3]]
actual = transpose([[1, -1], [2, 3]])
assert expected == list(map(list, actual))
```

Задание 3. (0.4 балла). Реализуйте функцию **scalar_product**, которая считает скалярное произведение двух iterable.

Элементы могут иметь тип *int* или *float*, а также быть строками. Строки могут быть либо представлением целых чисел (в том числе в двоичной или шестнадцатиричной системе счисления – используйте *built-in* функцию **int**), либо состоять из букв. Обработайте этот случай с помощью исключений, результатом вычисления в таком случае считайте **None**.

Воспользуйтесь функциями из модуля **itertools** и *built-in* функциями. Использовать циклы не разрешается.

Функцию сохраните в файле *functional.py*.

Пример

```
expected = 1
actual = scalar_product([1, '2'], [-1, 1])
assert expected == actual

actual = scalar_product([1, 'abc'], [-1, 1])
assert actual is None
```

Задание 4. (0.8 балла). Реализуйте функцию-генератор **flatten**, которая разворачивает вложенные итерируемые объекты в один iterable. Использовать `yield from` в Python 3 нельзя.

Обратите внимание, что строки, хоть они и являются итерируемыми, распаковывать не нужно. Постарайтесь также не вызывать функцию рекурсивно.

Функцию сохраните в файле *functional.py*.

Пример

```
expected = [1, 2, 0, 1, 1, 2, 1, 'ab']
actual = flatten([1, 2, xrange(2), [[], [1], [[2]]], (x for x in [1]), 'ab'])
assert expected == list(actual)
```

Задание 5. (0.3 балла). Напишите функцию **walk_files**, которая принимает в качестве аргумента некоторый путь к директории в файловой системе и возвращает дерево файлов в этой

папке в виде вложенных друг в друга словарей (изучите функции модулей **os** – в Python 2 и Python 3 – и **os.path** – в Python 2 и Python 3).

Ключи словаря – имена папок, а значения – списки содержащихся в них файлов и папок (папки представляются как такие же словари – рекурсивно). Допустимо другое представление.

Проверьте работу реализованной ранее функции **flatten** на результате работы.

Функцию также сохраните в файле *functional.py*.

Задание 6. (0.9 балла). Реализуйте декоратор **handle_error** и контекстный менеджер – назовите его **handle_error_context**,– которые позволяют обрабатывать ошибки в зависимости от переданных параметров:

- re_raise – флаг, отвечающий за то, будет произведен проброс исключения (типы исключений для обработки заданы параметром exc_type – см. ниже) из блока/функции на уровень выше или нет (по умолчанию **True**)
- log_traceback – флаг, отвечающий за то, будет ли при возникновении исключения типа exc_type отображен *traceback* (по умолчанию **True**)
- exc_type – параметр, принимающий либо отдельный тип, либо непустой кортеж типов исключений, которые должны быть обработаны (для всех остальных блока *except* не будет) – значение по умолчанию выставьте тип **Exception**

Для обработки и логирования *traceback* используйте функцию **sys.exc_info()** и схожие ей, модуль **traceback**, а логирование осуществляйте с помощью глобального для модуля объекта *logger* – **Logger**'а из стандартной библиотеки (модуль **logging**).

Обратите внимание, что при реализации декоратора и менеджера контекста код должен быть переиспользован – простого копирования требуется избежать.

Сохраните все в файле *error_handling.py*.

Пример 1

```
# log traceback, re-raise exception
with handle_error_context(log_traceback=True, exc_type=ValueError):
    raise ValueError()
```

Пример 2

```
# suppress exception, log traceback
@handle_error(re_raise=False)
def some_function():
    x = 1 / 0  # ZeroDivisionError

some_function()
print(1)  # line will be executed as exception is suppressed
```

Пример 3

```
# re-raise exception and doesn't log traceback as exc_type doesn't match
@handle_error(re_raise=False, exc_type=KeyError)
def some_function():
    x = 1 / 0  # ZeroDivisionError

some_function()
print(1)  # line won't be executed as exception is re-raised
```

Общие замечания ко всем заданиям

Тесты к решениям, как, впрочем, и полностью все задания делать не обязательно.